Representational State Transfer (REST) Principles  
   
The web architecture for REST can be described as a set of constraints applied to elements within the architecture.  
   
The Null Style is an empty set of constraints.  
   
----  
Client-Server constraints: Used to seperate the user interface concerns from the data storage concerns.  
Pros: Improves portability of UI across multiple platforms, Improves scalability by simplifying server components.  
   
----  
Client-Stateless-Server style looks to make communication be stateless. The request from client to server  
must contain all the info necessary to understand the request, can't use stored info on server.  
   
Pros: Improved visibility since it has all info in one single datum to determine full nature of the request,  
Improves reliability by making recovering from partial failures easier, and Improves scalability since  
no state is stored between requests, allowing resources to free and simplifying implementation.  
   
Cons: may decrease network performance by increasing repetitive data, also reduces server's control  
over consistent application behavior.  
   
----  
Cache constraint can be used to improve network efficiency. Requires that data within a response to a request  
be labeled as cacheable or non-cacheable. Alows client cache to reuse that response data for later, equivalent requests.  
   
Pros: Improves efficiency, scalability, and user-perceived performance. Cons: Decreased reliability if stale data is  
different than new data that's ignored.  
   
----  
Uniform interface between components allows simplifying and improved visibility of interactions. Cons: degrades efficiency  
since info is transferred in a standardized form.  
   
----  
Layered System restricts components to only see immediate layers they are interacting with. Pros: bounds system complexity  
and promotes substrate independence.  
   
----  
REST ignores details of component implementation and protocol syntax to focus on the roles of components,  
interaction constraints, and interpretation of significant data elements.  
   
Key aspect of REST is the nature and state of an architecture's data elements.  
Resource - the intended target of a hypertext reference  
Resource identifier - URL, URN  
Representation - HTML document, JPEG image  
Representation metadata - media type, last-modified time  
Resource metadata - source link, alternates, vary  
Control Data - if-modified-since, cache-control  
   
Any information that can be named can be a resource: a document/image, temporal service ("todays weather in LA"),  
a collection of other resources, a person, etc. If it's a target of a hypertext reference, must fit definition  
of a resource.  
   
A resource is a conceptual mapping to a set of entities. Resource Representations and/or identifiers are the  
values in the set. You can reference to an empty set (a concept before it's actually realized).  
   
-----------------------  
SIMPLIFIED VERSION  
   
REST - Representational State Transfer  
It's basically about using HTTP verbs: GET, POST, PUT, DELETE and HEAD  
These are used to act on resources represented by individual URIs.  
   
UNDERLYING PRINCIPLE  
Main idea is that you're working with the HTTP protocol. Most web service ideas only use GET and/or POST  
requests. Requests and responses of the HTTP protocol consist of a head and a body. Head is the envelope  
and body is the contents. REST uses URI, HTTP verbs, HTTP headers and HTTP response codes to communicate.  
   
SOAP generally uses the URI from the head and then sends another envelope in the body to communicate more info  
about the request to the server and the response to the client. Basically an envelope inside an envelope.  
   
HTTP VERBS AND THEIR MEANING  
   
GET  
Give me the current representation for the given URI. Headers can modify the behaviour slightly to minimize  
needless traffic and processing. Ex: using the If-Modified-Since to only show representation if changed since  
the given date.  
   
DELETE  
Deletes the representation at the given URI.  
   
HEAD  
Equivalent to a GET request but only returns the HEAD, omitting the body. Gets the meta data  
about a resource, aka checking if it exists.  
   
PUT  
Used to place a resource to a specific URI.  
   
POST  
Used to add a resource to the end of a list of resources when you don't know the URI in advance.  
Put is used to update it then.